MAST 680 Assignment 1

Serly Ishkhanian

February 12, 2023

Abstract

The purpose of this paper is to use Dynamical Mode Decomposition (DMD) technique on two video clips containing background and foreground objects to separate them into foreground and background videos. We will observe how we use Singular Value Decomposition (SVD) to compress the frames of the video streams to perform dimensionality reduction without losing much information and how we can apply DMD to separate the foreground and background.

1 Introduction and Overview

1.1 Introduction

Separation of foreground and background objects in video clips is a beneficial and crucial technique in many applications where recognizing, classifying, and tracking objects in videos is of interest. With numerous applications in the field of fluid dynamics, physics, biology, and many others, the Dynamical Mode Decomposition (DMD), a data-driven technique, can be applied to high dimensional complex data that produces spatial-temporal modes and a best-fit linear dynamical system that informs us how those modes change in time. DMD allows moving from one frame to another with simple matrix multiplication [1], and the modes of the DMD matrix can be used to separate the background and foreground objects in the video (discussed in Section 2.2).

1.2 Problem overview

Given the video clips of racing cars and a ski man going down a hill, can we separate the foreground objects, such as the racing cars and the ski man, from the static background? In this paper, we will observe how we can apply Singular Value Decomposition (SVD) to perform dimensionality reduction on the data to make it possible to operate with, while keeping as much important information as possible. Then, we will apply DMD and obtain the eigenvalues and eigenvectors (spatial-temporal modes) of the DMD matrix; this would allow us to distinguish between the fast and slow-moving objects in the video. We will also see how the static background can be represented by a low-rank matrix (because of its similarity over the frames), and the foreground objects can be represented by a sparse matrix. Theoretical backgrounds of the techniques used are in Section 2, and implementation and results of those techniques are in Sections 3 and 4, respectively.

2 Theoretical Background

A video is a sequence of frames where each frame is a two-dimensional data matrix. For each video, let $F \in \mathbb{R}^{M \times N}$ be the data matrix whose columns $X_i \in \mathbb{R}^{M \times 1}$ represent the vectorized form of the frames. This means that as we go across the columns of F, it will be as if we are moving through time frame by frame. M represents the number of pixels in each frame, and N represents the number of frames in the video.

$$F = \begin{bmatrix} X_1 & X_2 & X_3 & \dots & X_N \end{bmatrix}. \tag{1}$$

2.1 Singular Value Decomposition

Since F is a large matrix, operations using F can be computationally hard and time-consuming. Thus, we would like to find a low-rank approximation of F by applying SVD. However, it may not always be feasible to obtain the full SVD of enormous matrices because of computational complexity. So, we will compute the compact SVD. Assuming that rank(F)= r, we can compute the compact SVD [1]:

$$F = U_r \Sigma_r V_r^*, \tag{2}$$

where $U_r \in \mathbb{C}^{M\times r}$, $V_r \in \mathbb{C}^{N\times r}$, and $\Sigma_r \in \mathbb{R}^{r\times r}$ is a diagonal matrix whose entries are the ordered positive r singular values of F. Also, U_r, V_r are semi-unitary matrices such that $U_r^* U_r = V_r^* V_r = I_r$. What the compact SVD does is that it removes the unnecessary zero singular values by keeping the first r rows and column of Σ, and it removes the last $M - r$ and $N - r$ columns of U and V, respectively, of the full SVD.

In relation to our spatial-temporal data, the compact SVD decomposes F such that U_r represents the spatial component of the data, V_r represents the time component of the data, and the singular values in Σ_r retain important information about the data.

Lastly, using the first s columns of U_r, we can reduce the dimension of our data matrix F by projecting F onto the subspace spanned by the s columns of U_r. This process will still preserve as much as since the columns of U_r are ordered accordingly in order of importance of the singular values. This process will give us the reduced matrix F_s of size $s \times N$ as follows [1]: In Section 3, we will see how we can choose s.

$$F_s = U_s^* F \tag{3}$$

2.2 Dynamical Mode Decomposition

Once we obtain the low-rank approximated matrix F_s of F, we will create our data matrices $X, Y \in \mathbb{C}^{s\times N-1}$ [1]:

$$X = \begin{bmatrix} X_1 & X_2 & X_3 & \dots & X_{N-1} \end{bmatrix}, \quad Y = \begin{bmatrix} X_2 & X_3 & X_4 & \dots & X_N \end{bmatrix}, \tag{4}$$

where columns of X represent all the frames in the video from the first frame to before the last. Similarly, columns of Y represent the frames of the video starting from the second frame until the last, i.e. Y is the same data matrix but shifted one frame into the future such that $Y_i = X_{i+1}$ for all $1 \leq i \leq N - 1$.

Moving from one time frame to another could be a nonlinear mapping. So, is there a method of finding a linear mapping that enables us to do so? This is where DMD comes in; DMD attempts to find the best linear mapping, i.e a matrix $A \in \mathbb{C}^{s\times s}$, that allows us to move from X to Y such that we can express Y as follows [1]:

$$Y \approx AX. \tag{5}$$

Finding A leads to solving a minimization problem [1]:

$$\underset{A\in\mathbb{C}^{s\times s}}{\operatorname{argmin}} \quad \|Y - AX\|_F^2, \tag{6}$$

where it is of interest to find a matrix A so that AX is very close to Y. This closeness of matrices is measured by the Frobenious norm $\|\cdot\|_F$; which is the square root of the sum of the squares of the elements of a matrix. The exact solution of (8) is [1]:

$$A = YX^\dagger, \tag{7}$$

where $X^\dagger$ is the pseudoinverse of X.

Now, how can we use the DMD matrix to separate the background and foreground objects in the video? Obtaining the eigenvalues $\mu \in \mathbb{C}$ and their corresponding eigenvectors $\phi \in \mathbb{C}^{s\times 1}$ of the DMD matrix A will give us information regarding the underlying system; the eigenvectors are the spatial modes of the data representing the most dominant patterns in the data, and the eigenvalues correspond to discrete time dynamics that inform us how the modes change in time. Having this knowledge of the eigendecomposition of the DMD matrix will allow us to forecast how the system changes in the short term, and we can utilize the eigendecomposition to separate the background and foreground objects.

Separating the background and foreground is just separating the fast modes and slow modes of the DMD matrix. What does this mean? Considering the continuous time eigenvalues $\omega_i = \ln(\mu_i)/dt$, where dt = video duration/number of frames, the modes associated with the eigenvalues $\mid \omega_p \mid \approx 0$ are the slow modes (the background) since they don't change much in time and the remaining modes, associated with $\mid \omega_j \mid$ that are bounded away from zero for all $j \neq p$ where $p \in \{1, 2, \ldots, s\}$, represent the fast modes (the foreground objects) [1]. In Section 3, we will set a threshold $\lambda > 0$ to distinguish between the modes. DMD reconstruction of F_s is $X_{DMD} \in \mathbb{R}^{s\times N}$. Whose columns we can think of as the reconstructed frames [1]:

$$X_{DMD}(t) = b_p\phi_p e^{\omega_p t} + \sum_{j\neq p} b_j\phi_j e^{\omega_j t}, \tag{8}$$

where each b_i is the initial amplitude of each mode. So, the vector $b \in \mathbb{C}^s$ can be found by $b = \Phi^\dagger X_1$, where $\Phi \in \mathbb{C}^{s\times s}$ whose columns are ϕ_i [1], since we know that at $t = 0$,(8) is to X_1. Although each term of (8) is complex, they add up to a real-valued matrix. Thus, when dividing the DMD terms into its low-rank and sparse reconstructions, it is essential to be careful with the complex elements as it affects the accuracy. Consider the reconstruction of the approximate low-rank that corresponds to the background video [1]:

$$X_{Low-Rank} = b_p\phi_p e^{\omega_p t}, \tag{9}$$

and from the fact that $X = X_{Low-Rank} + X_{Sparse}$, we can obtain the DMD's approximate sparse approximation:

$$X_{Sparse} = \sum_{j\neq p} b_j\phi_j e^{\omega_j t}, \tag{10}$$

which can be calculated with real-valued elements:

$$X_{Sparse} = F_s - \mid X_{Low-Rank} \mid, \tag{11}$$

where $\mid \cdot \mid$ denotes the modulus of each element in the matrix. One issue that could arise is that (11) could produce negative value pixel intensities. To counteract this, a matrix $R \in \mathbb{R}^{s\times N}$ is created to contain those negative values and make the following adjustments to account for the magnitudes of the complex values from the DMD reconstruction:

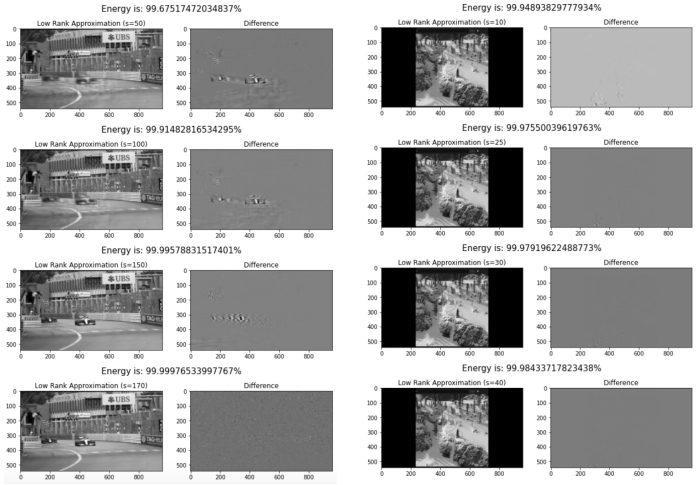
$$X_{Low-Rank} \mid \leftarrow R + \mid X_{Low-Rank} \mid, \quad \text{and} \quad X_{Sparse} \leftarrow X_{Sparse} - R. \tag{12}$$

This ensures that the approximate low-rank and sparse DMD matrices are real-valued [1].

3 Algorithm Implementation and Development

3.1 Setup

Before implementing the above techniques, we must load the data from both video files: monte_carlo_low.mp4 and ski_drop_low.mp4. Initially, both videos were 4D datasets. Converting them to greyscale transformed them into a 3D dataset. The Monte Carlo video contains 379 frames of size 540 by 960. The Ski Drop video has 454 frames of the same size. We then turn the datasets into a 2D matrix (as mentioned in Section 2 by vectorizing the frames so that the columns of each matrix represent the frames of the video. Thus, for both videos, $F \in \mathbb{R}^{518400\times 379}$ and $F \in \mathbb{R}^{518400\times 454}$, respectively.



(a) (b)

Figure 1: In (a), first frame of Monte Carlo video is used to test the low-rank approximation using different energy levels. In (b), 425^{th} frame of Ski Drop video is used (for the main reason that in this frame we can see the man at the bottom) to test the low-rank approximation using different energy levels. The right columns of (a) and (b) show the difference between the original frame and low-rank approximation.

3.2 Comppact SVD and Low Rank Approximation

Since this is a high-dimensional dataset, it comes with computational complexities; we encountered an issue of python crashing when attempting to compute the DMD matrix A. To produce a low-rank approximation of F, first, we get its compact SVD. Then, to decide the value of s, we compute the cumulative energy by summing the squares of the s singular values divided by the total sum of the squares of the singular values; it is a measure that tells us how much of the data can be explained by the s singular values see figure in Appendix C.

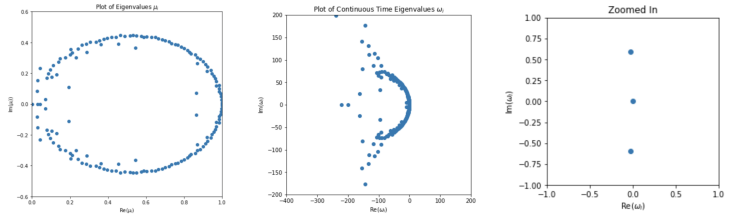
So, to choose the value of s, we observe how the low-rank approximated frames look in Figure 2 using different values of s and the difference between the original image and the low-rank approximation.

From Figure 1a, we We can see that the cumulative energy is 99.675% when $s = 50$, the approximation is not good. So, as we increase s (by also increasing the energy), we observe that the low-rank approximations improve. Using $s = 170$ with 99.99976% energy captured for Monte Carlo, we can get a satisfactory low-rank approximation of the original frame see Figure 6 in Appendix C. From Figure 1b, we can see that using only $s = 30$ with 99.979% energy captured for the Ski video is enough to produce a good low-rank approximation since as s increases, there is clearly not much changing. Now, the low rank approximation of our data matrix can be represented by $F_s = U_s^* F$ where $F_s \in \mathbb{R}^{s \times N}$.

3.3 Applying DMD

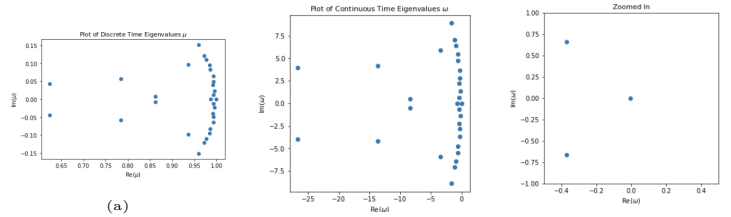
Now that we have our low-rank approximation matrix F_s, we will apply the DMD technique. As seen in the algorithm in Appendix B, we start by creating our X, and Y matrices, which would allow us to create the DMD matrix $A = YX^{\dagger}$. After achieving this, we compute the complex eigenvalues and eigenvectors of A.

To distinguish between the slow modes and fast modes, we set the value of the threshold λ by observing Figures 2 and 3 of the continuous time eigenvalues near 0 (as mentioned in Section 2.2). We can set $\lambda = 0.5$ as the cut off point since the points in both figures 2c and 3c are above 0.5. Using this threshold, we can distinguish between the slow modes and fast modes, and create our low-rank background and sparse foreground following the algorithm 1. For more



(a) (b) (c)

Figure 2: Plots of the discrete and continuous time eigenvalues of DMD matrix of Monte Carlo video. (c) shows the zoomed in version of (b) near 0.



(a) (b) (c)

Figure 3: Plots of the discrete and continuous time eigenvalues of DMD Matrix of Ski Drop video. (c) shows the zoomed in version of (b) near 0.

steps, please view the detailed algorithm in Appendix B.

Algorithm 1: DMD Process After Obtaining DMD Matrix A

Get eigenvalues and eigenvectors of A
Compute $b = \Phi^{\dagger} X_1$
Compute w_i for $i = 1, \ldots, s$
for $i = 1 : s$ **do**
  Get indices i such that $\mid \omega_i \mid < 0.5$
  Create a vector b_{slow} whose entries are all zeros except at the positions of those indices
  Set $b_{slow}[i] = b[i]$
end for
Create $X_{LowRank}$
Create $X_{Sparse} = F_s - X_{LowRank}$
Create matrix R whose entries are all zeros except the negative values of X_{Sparse}
Reassign $X_{Sparse} \leftarrow X_{Sparse} - R$
Reassign $X_{LowRank} \leftarrow \mid X_{LowRank} \mid + R$
Plot images of $X_{LowRank}$ and X_{Sparse}

4 Computational Results

In this section, we will present the results obtained from implementing the algorithm in Appendix B in Python to separate the foreground and background objects in the video by testing on a specific frame.

Figures 4 and 5 show the results of applying DMD for background and foreground object separation done on frame 1 of Monte Carlo video, and frame 425 of Ski Drop video. As we can see from Figure 5c, the algorithm was able to separate the ski man and some falling snow (foreground objects) from the background. Lastly, during the trial process, a few observations were made. Interestingly, for the Monte Carlo video, when R was added back to the modulus of $X_{LowRank}$ as shown in algorithm 1 the results were not good (see figure 8 in Appendix C). One of the cars was still visible in the background frame. The figure obtained in Figure 4b was the outcome

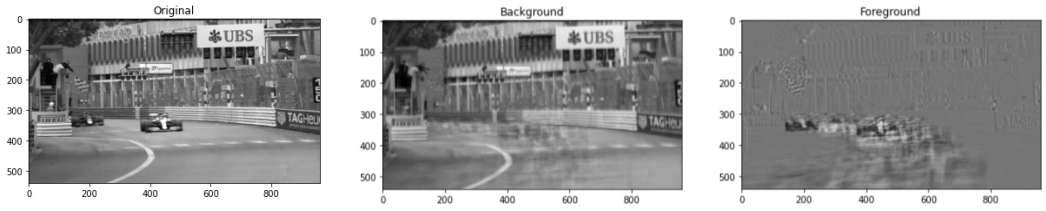


Figure 4: Visualization of (a) the original frame, (b) the background, and (c) the foreground objects in frame 1 of Monte Carlo Video as a result of applying DMD ($s = 170$).

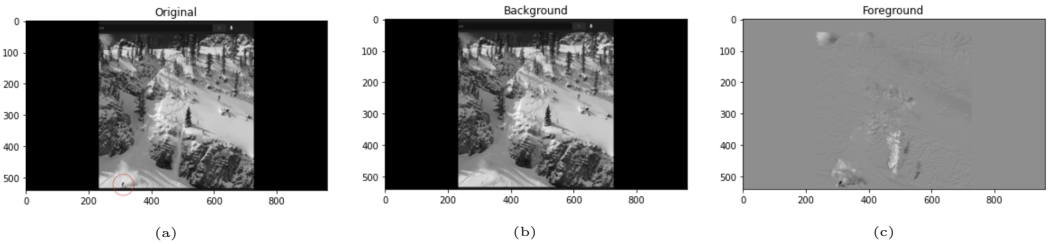


Figure 5: Visualization of (a) the original frame, (b) the background, and (c) the foreground objects in frame 425 of Ski Drop Video as a result of applying DMD ($s = 30$).

of adding R to $X_{LowRank}$, and then specifying to take the real values when plotting the image (see Appendix B). No such issue was observed in the separation process of the Ski Drop frame. Moreover, Figures 9 and 10 in Appendix C, show the results of the background and foreground separation process on the same frames using full energy ($s = N$). Surprisingly, the results are not much better. For Monte Carlo frame, there is slight white shadow on the background image. While for the Ski Drop frame, there is not much difference that can be seen between Figures 10a Appendix C and 5b.

5 Summary and Conclusions

In this report, we discussed the theoretical background of the methods used to perform background-foreground separation on two different video clips: The Monte Carlo video (6 seconds long) and the Ski Drop video (8 seconds long). We saw each greyscale version of those two videos converted into an enormous two-dimensional dataset. Since it is computationally hard to compute with such matrices, we obtained the compact SVD of our data matrices to reduce them to a lower dimension while preserving important information about the data. Then, after we created the data matrices X, Y, we computed the DMD matrix A and its eigenvalues and eigenvectors. Those tell us about the dominant patterns in the underlying dynamical system. We also observed how distinguishing between the slow and fast modes can lead to the separation of foreground and background objects, as seen from the results in Section 4.

References

[1] Jason J. Bramburger. *Data-Driven Methods for Dynamical Systems.* Concordia University, 2023.

Appendix A Python Functions

- `skvideo.io.vread(video.mp4)` returns a 4D array of the video.
- `skvideo.utils.rgb2gray(videodata)` computes the grayscale video from the input video returning the standardized shape (number of frames, height, width, 1).

- `reshape` function will reshape an array into the dimension you want. We use it to vectorize the frames.
- `u,s,vh = np.linalg.svd(M,full_matrices = False)` produces the compact SVD of the matrix M.
- `plt.imshow` takes a 2D array (in our case, a frame) and produces a picture of the frames.
- `sum` will add the numbers in an array.
- `np.delete(array,n,axis=1)` deletes the n-th column of array.
- `p.linalg.pinv(X)` computes the pseudoinverse of matrix X.
- `np.linalg.eig(A)` returns the eigenvalues and eigenvectors of matrix A.
- `np.abs(element)` will take the modulus of a complex element.

Appendix B Python Code

```
# Import
import numpy as np
import cv2
from matplotlib import pyplot as plt

# Install
pip install scikit-video
import skvideo.io

# Load the video (instert the path of video used)
videodata = skvideo.io.vread("/content/ski_drop_low.mp4")
number_of_frames, height, width, colors = videodata.shape
grey_ski = skvideo.utils.rgb2gray(videodata)

# Get matrix F of size M by N

frames = []

for frame in grey_ski:

    # Vectorize the frames by stacking the columns vertically
    reshaped_frame = frame.reshape((540*960))

    # This adds the frames into the matrix as rows
    frames.append(reshaped_frame)

# Transpose_frames is matrix F
Transpose_frames = np.array(frames).T

# Get compact SVD of F
u,s,vh = np.linalg.svd(Transpose_frames,full_matrices = False)


# Calculate energy of each singularvalue and the cumulative energy

total_squared_sum_sigmas = sum(i*i for i in s)
```

```
energy_of_each_singular_value = [(i*i)/total_squared_sum_sigmas for i in s]
k = [(i+1) for i in range(len(s))]


sings = 0
cumulatives =[] # contains cumulative energies

for i in s:
  sings += i*i
  cum = sings/total_squared_sum_sigmas
  cumulatives.append(cum)



# Function to construct the low-rank images that would help in deciding whats the best s
# Argument is the number of singular values desired
# It also outputs the cumulative energy

def low_rank_image_reconstruction(number_of_singularvalues):
  new_matrix = u[:,0:number_of_singularvalues].conj().T @ Transpose_frames
  f1 = u[:,0:number_of_singularvalues]@new_matrix

  # Create the energy
  energy_used = cumulatives[number_of_singularvalues-1]*100



  # Extract the first frame
  reduced_frame = np.reshape(f1[:,425],(height,width))

  fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(10,8))
  ax1.imshow(reduced_frame,cmap='gray')
  ax1.set_title("Low Rank Approximation (s="+str(number_of_singularvalues)+")")
  ax2.imshow(np.reshape(Transpose_frames[:,425]-f1[:,425],(height,width)),cmap='gray')
  ax2.set_title(" Difference ")

  fig.suptitle("Energy is: "+str(energy_used)+"%",y=0.7,fontsize=15)
  plt.show()


# Line Plot of Cumulative energy vs singularvalues k
plt.plot(k,cumulatives)
plt.xlabel("K Singular Values ")
plt.ylabel("Energy")
plt.title("Plot of Cumulative Energy (Ski Drop Video)")


# Produce the low-rank approximated pictures and the difference
low_rank_image_reconstruction(10)
low_rank_image_reconstruction(25)
low_rank_image_reconstruction(30)
low_rank_image_reconstruction(40)


# Create matrix F_s (denoted as new_videodata)
```

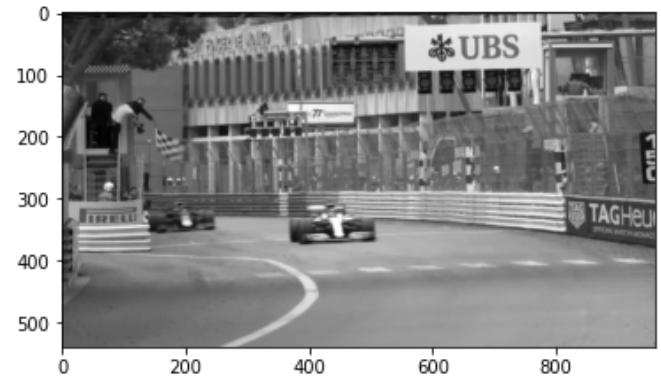
```
num_of_sings = 30       #specify number of singularvalues
new_videodata = u[:,0:num_of_sings].conj().T @ Transpose_frames


# Create matrix X and Y
X = np.delete(new_videodata,453,axis=1)
Y = np.delete(new_videodata,0,axis=1)


# DMD matrix A
A = Y @ np.linalg.pinv(X)


# Get eigenvalues and eigenvectors of A
evalues, evectors = np.linalg.eig(A)


# Create omega: continuous time eigenvalues of A
omega =[] # set of transformed eigenvalues

dt =8/number_of_frames #cduration/number of frames

for eigenvalue in evalues:
  omegak = np.log(eigenvalue)/dt
  omega.append(omegak)

omega = np.array(omega)


# Plot the discrete time eigenvalues
plt.scatter(evalues.real,evalues.imag)
plt.xlabel("Re($\mu$)")
plt.ylabel("Im($\mu$)")
plt.title("Discrete Time Eigenvalues $\mu$ (Ski Drop Video)")



# Find coefficients of the eigendecomposition
b = np.linalg.pinv(evectors) @ X[:,0]


# Apply DMD

# Find the indexes of |omega|< 0.5
slow_index = []
for i in range(len(omega)):
  if abs(omega[i])<0.5:
    slow_index.append(i)

# Create the coefficients b corresponding to the slow modes
b_slow = np.zeros(b.shape).astype(complex)

for i in slow_index:
  b_slow[i] = b[i]
```

```
# Construct the X_lowrank by eigendecompsoition
slow_x = np.zeros((num_of_sings,number_of_frames)).astype('complex')

for i in range(number_of_frames):
  for j in range(num_of_sings):
    if b_slow[j]!=0:
      slow_x[:,i] = slow_x[:,i]+(b[j]* np.exp(omega[j]*i))*evectors[:,j]


# Construct X_sparse
x_sparse = new_videodata - np.abs(slow_x)

# Construct the matrix R
new_r =np.zeros(x_sparse.shape)

for i in range(number_of_frames):
  for j in range(num_of_sings):
    if x_sparse[j,i].real <0:
      new_r[j,i] = x_sparse[j,i]

# Subtract R from x_sparse
new_sparse = x_sparse - new_r

# Add R to x_lowrank
new_slow =np.abs(slow_x) + new_r

#### NOTE: for Monte Carlo video, the above step didn't work nicely. Instead, I did :####
new_slow = slow_x + new_r

#Plotted new_slow (the background) by specifying .real :
plt.imshow(np.reshape((u[:,0:num_of_sings]@new_slow[:,0]).real,(height, width)),cmap = 'gray')


### Plotting the background and foreground ###
plt.imshow(np.reshape((u[:,0:num_of_sings]@new_slow[:,425]).real,(height, width)),cmap = 'gray')
plt.title("Background")

plt.imshow(np.reshape((u[:,0:num_of_sings]@new_sparse[:,425]).real,(height, width)),cmap = 'gray'
plt.title("Foreground")
#NOTE: The number 425 denotes the 425-th frame that I want to use to perform the separation. It c
```

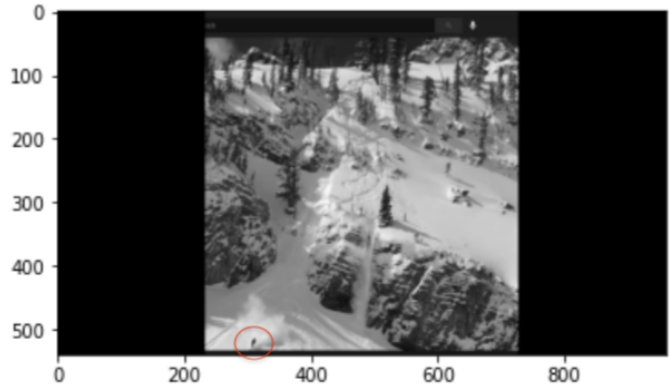
Appendix C Additional Plots

- Figure 6 demonstrates the original pictures of the frames of (a) Monte Carlo and (b) Ski Drop.
- Figure 7 shows the cumulative energies of the singular values for both Monte Carlo and Ski Drop datasets.

- Figure 8 represents the background image of frame 1 of Monte Carlo video when computing the step: $X_{LowRank} \leftarrow R+ \mid X_{LowRank} \mid$.
- Figures 9 and 10 show the results of the DMD process to separate the background and foreground on the same frames, but using full energy ($s = N$).

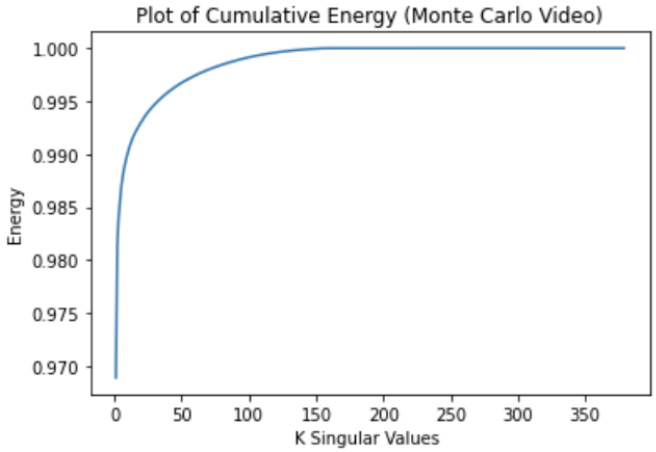


(a) Monte Carlo Frame 1

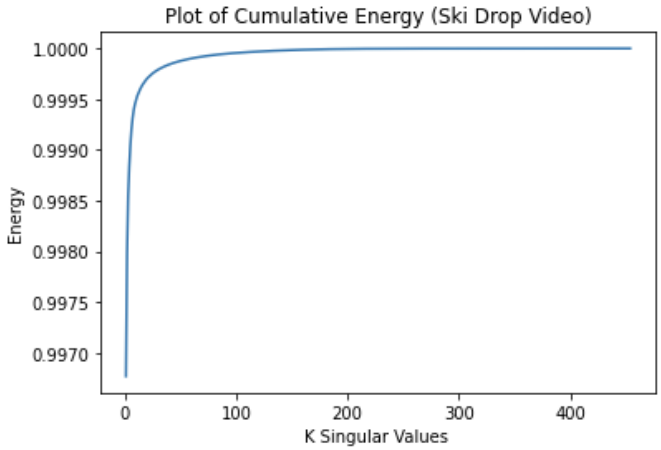


(b) Ski Drop Video Frame 425

Figure 6: Original frame images of Monte Carlo video and Ski Drop video.



(a)



(b)

Figure 7: Plots of the cumulative energies for both (a) Monte Carlo dataset and (b) Ski Drop dataset.

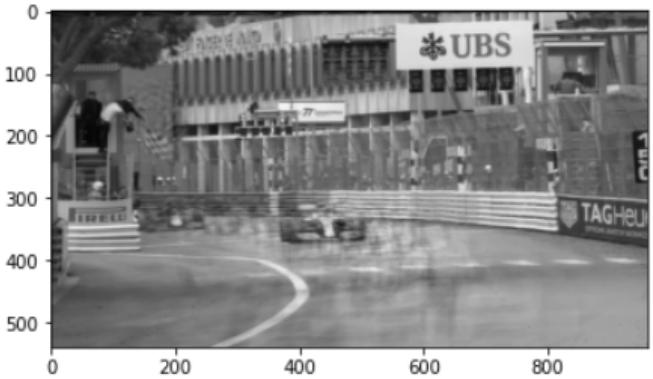
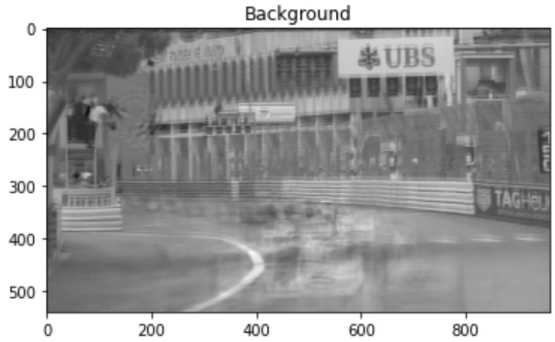
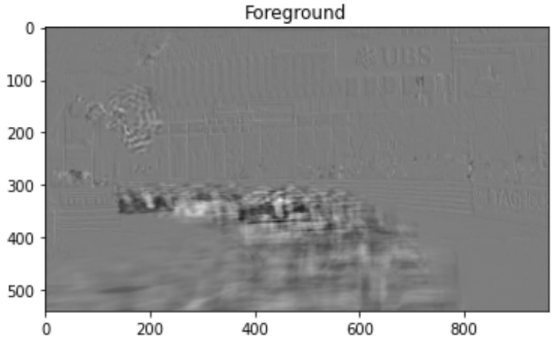


Figure 8: Background image resulting from adding R to the modulus of $X_{LowRank}$.

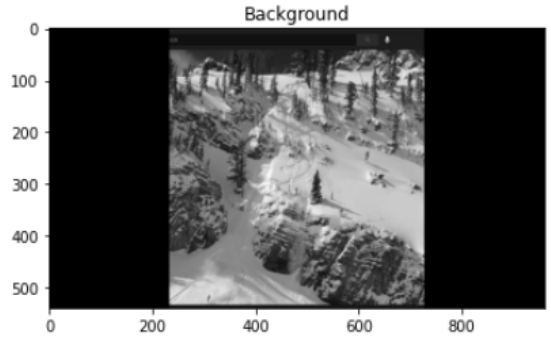


(a)

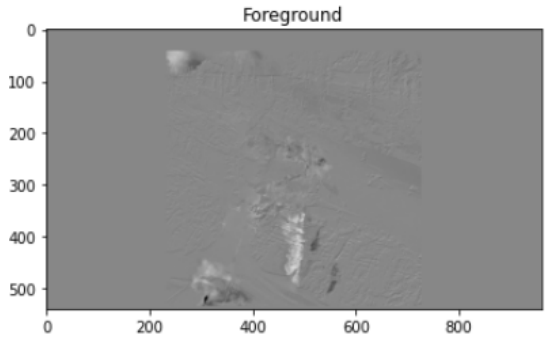


(b)

Figure 9: Result of background-foreground separation of Monte Carlo frame 1 when using full energy.



(a)



(b)

Figure 10: Result of background-foreground separation of Ski Drop of frame 425 when using full energy.